CLAIMS

1.   A method comprising:

    a. determining when to generate a dump file; and

    b. generating a dump file by gathering at least:

        i.   thread information for at least one running thread,

        ii. context information for the thread,

        iii. callstack information for the thread,

        iv. process information for the process in which the thread is running, and

        v. information identifying a reason for generating the dump file.

2.   The method as recited in Claim 1, wherein generating the dump file further includes storing the dump file to a storage medium.

3.   The method as recited in Claim 1, wherein generating the dump file further includes gathering processor information about at least one processor.

4.   The method as recited in Claim 1, wherein determining when to generate the dump file further includes determining that an exception has occurred.

5.   The method as recited in Claim 4, wherein the dump file does not further include any significant portion of a dynamically allocated memory.

6.   The method as recited in Claim 5 wherein the dump file does not include any portion of a global initialized or uninitialized memory.

7.   The method as recited Claim 5 wherein the dump file does not include any portion of the executable instructions used by the processor to execute the program.

8.   The method as recited in Claim 1, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered the exception.

9.   The method as recited in Claim 1, wherein the callstack information is a kernel stack.

10.   The method as recited in Claim 1, wherein the process information identifies the process that initiated the thread.

11.   The method as recited in Claim 1, further comprising:

a. allocating a buffer space in memory during an initialization process; and

b. reserving space on a storage medium drive suitable for writing the contents of the buffer.

12. The method as recited in Claim 11, wherein:

a. generating the dump file further includes initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a dump file; and

b. upon system re-initialization, transferring the dump file from the storage medium to at least one external computer.

13. The method as recited in Claim 12, further comprising upon re-initialization, after having stored the dump file to the storage medium, accessing the dump file on the storage medium and using at least a portion of the dump file to further understand an exception that was at least one reason for generating the dump file.

14. The method as recited in Claim 1, wherein the dump file is a user minidump file associated with at least one non-operating system program.

15. The method as recited in Claim 1, wherein generating the dump file further includes gathering callstack information for all running threads.

16. The method as recited in Claim 1, wherein the callstack information is a user callstack.

17. The method as recited in Claim 1, wherein generating the dump file further includes gathering processor context information for all running threads.

18. The method as recited in Claim 1, wherein generating the dump file further includes gathering a listing of all loaded modules for the faulting application program.

19. The method as recited in Claim 1, wherein the dump file is a directory indexed file that uses relative virtual addresses (RVAs).

20. A computer-readable medium having computer-executable instructions for performing steps comprising:

a. determining when to generate a dump file; and

b. generating a dump file by gathering at least:

    i. thread information for at least one running thread,

    ii. context information for the thread,

    iii. callstack information for the thread,

iv. process information for the process in which the thread is running, and

v. information identifying a reason for generating the dump file.

21. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes storing the dump file to a storage medium.

22. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes gathering processor information about at least one processor.

23. The computer-readable medium as recited in Claim 20, wherein determining when to generate the dump file further includes determining that an exception has occurred.

24. The computer-readable medium as recited in Claim 23, wherein the dump file does not further include any significant portion of a dynamically allocated memory.

25. The computer-readable medium as recited in Claim 24 wherein the dump file does not include any portion of a global initialized or uninitialized memory.

26.   The computer-readable medium as recited Claim 24 wherein the dump file does not include any portion of the executable instructions used by the processor to execute the program.

27.   The computer-readable medium as recited in Claim 20, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread which encountered the exception.

28.   The computer-readable medium as recited in Claim 20, wherein the callstack information is a kernel stack.

29.   The computer-readable medium as recited in Claim 20, wherein the process information identifies the process that initiated the thread.

30.   The computer-readable medium as recited in Claim 20, further comprising computer-executable instructions for performing steps of:

        allocating a buffer space in memory during an initialization process; and

        reserving space on a storage medium drive suitable for writing the contents of the buffer.

31.   The computer-readable medium as recited in Claim 30, wherein:

        generating the dump file further includes initially storing the thread information, the context information, the callstack information, the

process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a dump file, and upon system re-initialization, transferring the dump file from the storage medium to at least one external different computer.

32. The computer-readable medium as recited in Claim 31, further comprising computer-executable instructions for performing steps of, upon re-initialization after having stored the dump file to the storage medium, accessing the dump file on the storage medium and using at least a portion of the dump file to further understand an exception that was at least one reason for generating the dump file.

33. The computer-readable medium as recited in Claim 20, wherein the dump file is a user minidump file associated with at least one non-operating system program.

34. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes gathering callstack information for all running threads.

35. The computer-readable medium as recited in Claim 20, wherein the callstack information is a user callstack.

36. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes gathering processor context information for all running threads.

37. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes gathering a listing of all loaded modules for the faulting application program.

38. The computer-readable medium as recited in Claim 20, wherein the .dump file is a directory indexed file that uses relative virtual addresses (RVAs).

39. An arrangement comprising memory, a data storage drive configured to write data files to at least one data storage medium, and a processor operatively coupled to the memory and the data storage drive and configured to:

    a. determine when to generate a dump file; and

    b. generate a dump file by gathering at least:

       i. thread information for at least one running thread,

       ii. context information for the thread,

       iii. callstack information for the thread,

       iv. process information for the process in which the thread is running, and information identifying a reason for generating the dump file.

40. The arrangement as recited in Claim 39, wherein generating the dump file further includes storing the dump file to a storage medium.

41. The arrangement as recited in Claim 39, wherein generating the dump file further includes gathering processor information about at least one processor.

42. The arrangement as recited in Claim 39, wherein determining when to generate the dump file further includes determining that an exception has occurred.

43. The arrangement as recited in Claim 43, wherein the dump file does not further include any significant portion of a dynamically allocated memory.

44. The arrangement as recited in Claim 43 wherein the dump file does not include any portion of a global initialized or uninitialized memory.

45. The arrangement as recited Claim 43 wherein the dump file does not include any portion of the executable instructions used by the processor to execute the program.

46. The arrangement as recited in Claim 39, wherein the dump file is a kernel minidump file associated with an operating system and the at

least one running thread is the single thread which encountered the exception.

47. The arrangement as recited in Claim 39, wherein the callstack information is a kernel stack.

48. The arrangement as recited in Claim 39, wherein the process information identifies the process that initiated the thread.

49. The arrangement as recited in Claim 39, further comprising computer-executable instructions for performing steps of:

allocating a buffer space in memory during an initialization process; and

reserving space on a storage medium drive suitable for writing the contents of the buffer.

50. The arrangement as recited in Claim 49, wherein:

generating the dump file further includes initially storing the thread information, the context information, the callstack information, the process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a dump file; and

upon system re-initialization, transferring the dump file from the storage medium to at least one external computer.

51.   The arrangement as recited in Claim 50, further comprising computer-executable instructions for performing steps of, upon re-initialization after having stored the dump file to the storage medium, accessing the dump file on the storage medium and using at least a portion of the dump file to further understand an exception that was at least one reason for generating the dump file.

52.   The arrangement as recited in Claim 39, wherein the dump file is a user minidump file associated with at least one non-operating system program.

53.   The arrangement as recited in Claim 39, wherein generating the dump file further includes gathering callstack information for all running threads.

54.   The arrangement as recited in Claim 39, wherein the callstack information is a user callstack.

55.   The arrangement as recited in Claim 39, wherein generating the dump file further includes gathering processor context information for all running threads.

56.   The arrangement as recited in Claim 39, wherein generating the dump file further includes gathering a listing of all loaded modules for the faulting application program.

57.   The arrangement as recited in Claim 39, wherein the dump file is a directory indexed file that uses relative virtual addresses (RVAs).

58.   A method for generating a minimal dump file, the method comprising:

   a.   detecting an exception; and

   b.   outputting:

      i.   information on a faulting thread and an associated process, and

      ii.   a list of loaded modules.

59.   The method as recited in Claim 58, further comprising storing the minimal dump file to a storage medium.

60.   The method as recited in Claim 58, further comprising transporting the minimal dump file using a communication resource.

61.   A method of communicating between a client process and a server process in a distributed processing system, comprising:

   a.   issuing, by the client process, a write dump file call having a plurality of call parameters comprising a process handle, a process identifier, a handle to a file where dump file information is to be written, and a dump type identifier;

   b.   receiving, by the server process, the write dump file call and parsing the call to retrieve the parameters; and

c. issuing, by the server process, a write dump file call acknowledgment providing a true-false indication.

62. The method as recited in Claim 61, wherein the plurality of call parameters further includes a pointer to a structure describing an exception in the client that caused the dump file to be generated.

63. The method as recited in Claim 61, wherein the plurality of call parameters further includes a pointer to an array of user data entry structures.

64. The method as recited in Claim 61, wherein the plurality of call parameters further includes a pointer to a dump file callback data pointer.

65. A method of communicating between a client process and a server process in a distributed processing system, comprising:

    a. issuing, by the client process, a read dump file call having a plurality of call parameters comprising a header of a dump file and a data type identifier of data to read from a dump file;

    b. receiving, by the server process, the read dump file call and parsing the call to retrieve the parameters; and

    c. issuing, by the server process, a read dump file call acknowledgment providing a true-false indication and a plurality of call return parameters comprising a pointer to a beginning of a

dump stream, and a stream size identifying the size of the dump stream.

66. The method as recited in Claim 65, wherein the plurality of call return parameters further includes a pointer to a dump file directory.